MASTER COPY KEEP THIS COPY FO[illegible] REPRODUCTION PURPOSES

AD-A226 203

REPORT DOCUMENTATION PAGE

Form Approved
OMB No 0704-0188

2

1. AGENCY USE ONLY (*Leave blank*)	2. REPORT DATE 1990	3. REPORT TYPE AND DATES COVERED Final 1 Apr 87 - 31 Mar 90

4. TITLE AND SUBTITLE

Algorithms and Architectures for High Speed Signal Processing

5. FUNDING NUMBERS

DAAL03-87-K-0033

AUTHOR(S)

Thomas Kailath

DTIC
ELECTE
AUG 31 1990
S B D

PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Stanford University
Stanford, CA 94305

8. PERFORMING ORGANIZATION REPORT NUMBER

SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

U. S. Army Research Office
P. O. Box 12211
Research Triangle Park, NC 27709-2211

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

ARO 24954.33-MA-SDI

11. SUPPLEMENTARY NOTES

The view, opinions and/or findings contained in this report are those of the author(s) and should not be construed as an official Department of the Army position, policy, or decision, unless so designated by other documentation.

12a. DISTRIBUTION/AVAILABILITY STATEMENT

Approved for public release; distribution unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT (*Maximum 200 words*)

The major results are in two areas: --

1. Studies of systematic design procedures for a class of structured algorithms often encountered in signal processing applications. These are being called Regular Iterative Algorithms (RIAs). These ideas have been successfully used by a former student who helped to develop this theory, Dr. S. K. Rao of AT&T Bell Laboratories in Holmdel, N.J. Dr. Rao has found the RIA results helpful in designing several fast integrated circuit chips for communications and signal processing applicaitons, some of which are being used in the AT&T - ZENITH joint effort on High Definition Television (HDTV).

2. The other set of results deals with the issue of designing configurable and fault-tolerant processor arrays such that if some of the processors

(Continued on reverse side)

14. SUBJECT TERMS Algorithms, Architectures, High Speed Signal Processing, Signal Processing Applications, Design Procedures	15. NUMBER OF PAGES 28
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL

in the given array are faulty, then a fault free array can be constructed comprising only the healthy processors. Such studies can be easily motivated in the case of Wafer Scale Integration (WSI) technology where for example, a large number of processors, configured in the form of a grid, can be put on a single wafer. Due to yield problems, some of the processors are invariably going to be faulty. In such a case, instead of treating the whole wafer as defective, one can work around the faulty processors and reconfigure the rest in the form of a grid. Thus, reconfiguration methodologies can be viewed as possible tools to increase the effective yield of the processing technology. The general models that have been explored consist of a set of identical processors embedded in a flexible interconnection structure that is configured in the form of a rectangular grid. In particular, models were studied that use only limited hardware resources (such as a single-track or only a few tracks along every grid line) and the first known efficient algorithms for reconfiguration in such models were developed. In the process new models were developed that use limited hardware and yet has higher reconfigurability than other models studied in the literature.

ARO 24954.3-MA-SDI

ALGORITHMS AND ARCHITECTURES FOR HIGH SPEED SIGNAL PROCESSING

FINAL REPORT

APRIL 1, 1987 -- MARCH 31, 1990

Professor Thomas Kailath
Information Systems Laboratory
Department of Electrical Engineering
Stanford University
Stanford, CA 94305

U.S. Army Research Office

24954-MA-SDI
DAAL03-87-K-0033

Table of Contents

1 Introduction

This is the final report on our work under ARO-SDI Contract DAAL03-87-K-0033, April 1, 1987 through March 31, 1990.

The major results are in two areas:

1. Studies of systematic design procedures for a class of structured algorithms often encountered in signal processing applications. These are what we have called Regular Iterative Algorithms (RIAs) for which our results are summarized in Section 2.

   It might be mentioned that these ideas have been successfully used by one of our former students who helped to develop this theory, Dr. S. K. Rao of AT&T Bell Laboratories in Holmdel, N.J. Dr. Rao has found the RIA results helpful in designing several fast integrated circuit chips for communications and signal processing applications, some of which are being used in the AT&T – ZENITH joint effort on High Definition Television (HDTV).

2. The other set of results deals with the issue of designing *configurable and fault-tolerant* processor arrays such that if some of the processors in the given array are faulty, then a fault free array can be constructed comprising only the healthy processors. Such studies can be easily motivated in the case of Wafer Scale Integration (WSI) technology where for example, a large number of processors, configured in the form of a grid, can be put on a single wafer. Due to yield problems, some of the processors are invariably going to be faulty. In such a case, instead of treating the whole wafer as defective, one can work around the faulty processors and reconfigure the rest in the form of a grid. Thus, reconfiguration methodologies can be viewed as possible tools to increase the effective yield of the processing technology. The general models that we have explored consist of a set of identical processors embedded in a flexible interconnection structure that is configured in the form of a *rectangular grid.* In particular, we studied models that use only limited hardware resources (such as a single-track or only a few tracks along every grid line) and developed the first known efficient algorithms for reconfiguration in such models. In the process we have also developed new models that use limited hardware and yet has higher reconfigurability than other models studied in the literature. Our results on this topic are summarized in Section 3.

2 Summary of Our Work on Regular Iterative Algorithms

Our previous work has shown that (see *e.g.* , [9, 10, 26, 27, 28, 29, 37]) that once a Regular Iterative Algorithm is designed for a given problem, then one can use the systematic design theory developed by us to generate efficient processor arrays. However, the following two important issues were left unresolved in the general area of designing special purpose processor arrays: 1. Systematic procedures for scheduling and mapping any given RIA was not fully developed. Our previous work had answered this issue partially and some theoretical gaps had remained in the framework 2. Most algorithms are not given to the designer in the RIA form and most initial representations are either sequential in nature (*e.g.*, FORTRAN or PASCAL programs) or general mathematical expressions. Hence, one needs to develop systematic procedures for deriving RIAs. In our work we have resolved both the above issues. In particular, we have developed a general framework for scheduling and mapping any given RIA; we have also developed a formal methodology for systematic formulation of RIAs starting from representations that we refer to as linearly indexed Assignment Codes. It can be shown that such codes are very close to the mathematical expressions of a wide variety of problems, especially in signal processing and matrix algebra.

In this section, we shall first briefly introduce RIAs and summarize our contributions in the analysis and implementation of such algorithms. We shall then briefly summarize our formal methodologies for scheduling and general RIA and also for deriving RIAs starting from general representations such as mathematical formulas.

2.1 Regular Iterative Algorithms and Our Contributions

A formal definition of RIAs can be found in [13, 29, 37]; here we shall introduce RIAs via a simple example.

Example 2.1 (2-D Filtering Algorithm): It can be shown (see [26, 29]) that certain numerically stable 2-D filtering algorithms due to *Deprettere and Dewilde* [5], Vaidyanathan and Mitra [43], and Fettweis [6], can all be written in the form:

For all (i, j, k), **where** $0 \leq i \leq n$ and $0 \leq j, k \leq N$, **do**

$$x(i, j+1, k+1) = f_{x,i}(x(i,j,k), y(i,j,k), w(i,j,k))$$

$$y(i+1, j, k) = f_{y,i}(x(i,j,k), y(i,j,k), w(i,j,k))$$

$$w(i-1, j, k) = f_{w,i}(x(i,j,k), w(i,j,k))$$

where $f_{x,i}$, $f_{y,i}$, $f_{w,i}$ are linear functions that are determined by a synthesis procedure.

□

The example displays the following (characteristic) features of an RIA:

Each variable in the RIA is identified by a label (*e.g.*, x, y or w in example 1) and an *index vector* (*e.g.* , $I = [i\ j\ k]^T$, in example 1). The set of all index points over which the variables of the RIA are defined is called the *index space*, which is a subset of the an S-dimensional integer lattice, Z^S.

The dependences among the variables are regular with respect to the index points. That is, if $x_1(I)$ is computed using the value of $x_2(I - \mathbf{d}_{12})$ then the *index displacement vector* $\mathbf{d}_{12}$, corresponding to this direct dependence, is the same regardless of the index point I.

The set of computations performed at every index point is often referred to as the *iteration unit* of the RIA. Also, note that although the direct dependences among the variables in an RIA are required to be independent of the index points, the actual computations carried out to evaluate these variables can depend on the index point. In general, the index space **I** will be semi-infinite along certain coordinates and bounded along others. The bounds on the coordinates will be referred to as the *size parameters* of the RIA.

The regular dependences of an RIA lead to a dependence graph with an iterative structure, which can be clearly demonstrated by embedding the dependence graph within the index space. That is, a set of V nodes is defined at every index point I in the index space **I**, where the i^{th} node represents the variable $x_i(I)$ in the RIA. As first noted by Karp *et al.* [13] and by Waite [45], the regularity of the dependence graph of an RIA can be concisely expressed in terms of a simpler and smaller graph called the *Reduced Dependence Graph* (RDG). The RDG of an RIA (see Fig. 1) has one node for each of the indexed variables in the RIA; it has a directed arc from node x_i to node x_j, if $x_j(I)$ is computed using the value of $x_i(I - \mathbf{d}_{ij})$ for some $\mathbf{d}_{ij}$; finally, each directed arc is assigned a vector weight representing the disnlacement of the index point across the direct dependence. We should note that the RDG and a specification of the index space **I**, completely characterize the dependence graph of an RIA; hence, the analysis of parallelism in an RIA is based on the analysis of the RDG instead of the larger dependence graph.

Some of our important results are enumerated below; for a detailed account of the work reported here previous work please see [26, 37]

1. A formal definition of systolic arrays was obtained that captured their generally accepted properties, especially regularity (mostly identical processors), spatial locality (local interconnections), temporal locality (no delay-free operations, or more precisely, all combinational elements are latched) and pipelined operation (throughput independent of the order, suitably defined, of the system). Some authors (*e.g,* Leiserson *et al.* [19]) had used only a subset of these properties, but the consensus in the literature appeared to have required all those mentioned above (see *e.g.,* [28] and [16]).

2. A reasonable generalization of the concept of systolic arrays that allowed implementation of a larger class of algorithms (including of course all systolic algorithms) was also developed. The generalization allowed the presence of register pipelines of various lengths at different points in a regular array of (mostly) identical processors, and sometimes also some LIFO (Last-In-First-Out) buffers. Such architectures have almost all the advantages that make systolic arrays so

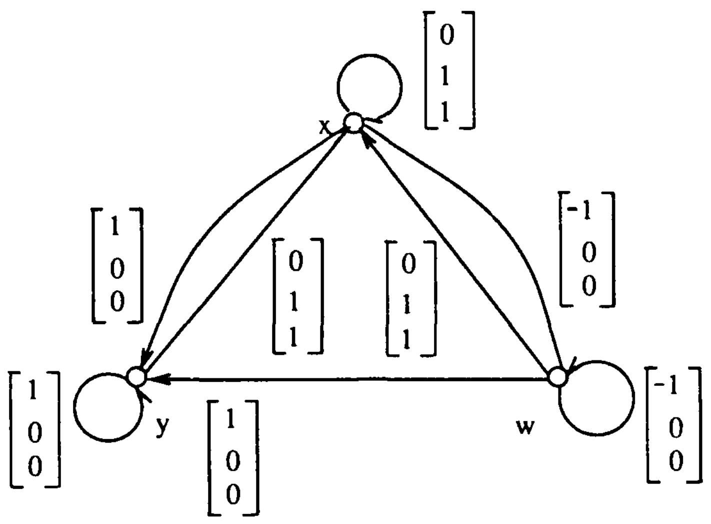


Figure 1: The RDG of the RIA in the above example.

appealing for VLSI; the only added requirement is that some of the processors may require certain amount of memory in them. We should note here that the memory requirement is not a major bottleneck, and certain commercial products such as the WARP developed at CMU, routinely provide such on-processor memory.

Rao *et al.* called such arrays Regular Iterative Arrays, and algorithms implementable on such arrays were dubbed as Regular Iterative Algorithms. It is convenient to use the acronym RIA to stand for either of these concepts, the exact one to be inferred from the context. Using the above concepts, and their consequences, one can show for example that there are Regular Iterative Algorithms (*e.g.* , RIAs for certain classes of 2-D filtering algorithms, RIAs for certain pivoting algorithms [37, 31, 34] *etc.*) that cannot be implemented on systolic arrays, as formally defined, but can be implemented on regular iterative arrays.

3. It was also shown [10, 16, 26, 37] that many algorithms in digital filtering (convolution, correlation, autoregressive, and moving-average filtering), numerical linear algebra, discrete methods for PDEs and ODEs, graph theory (transitive closure, some coloring problems) can be reformulated as RIAs. Systematic procedures for converting algorithms into RIAs, however, remained as an open problem.

4. For any RIA, formal methods to determine lower bounds on I/O latency and memory requirements were developed; systematic procedures for implementing *most RIAs* on regular processor arrays that can achieve the lower bound on I/O latency were also proposed (see

[26, 27, 29, 37]). We should mention here that these formal mapping techniques can generate several possible architectures, though in practice one stops once a few efficient (*i.e.* , meeting certain performance lower bounds) arrays have been obtained.

The theoretical question of scheduling and mapping any given RIA was left as an open problem. Our recent results on this topic are summarized in Section 2.1.

5. In the design of systolic arrays, several issues such as systematic procedures for designing *multi-rate* systolic arrays were resolved. In the conventional systolic array designs all operations were assumed to take the same amount of time; this led to unrealistic and inefficient design. Our design procedure allows one to carry out the design with more realistic processor modules that can increase the throughput by exploiting the fact that the time required to carry out different operations is generally different.

2.2 Scheduling and Implementing A Given RIA

Herein we describe our novel *subspace scheduling* scheme that can be used to schedule and implement any given RIA. Based on the analysis of the RDG, Karp *et al.* [13] showed that for a certain subclass of RIAs, one can always determine a scheduling vector Λ and scalars γ_{x_i}, such that a variable $x_i(I)$ can be scheduled at step $\Lambda^T I + \gamma_{x_i}$. Such schedules will be called *uniform affine schedules.* Thus, two variables $x_i(I)$ and $x_i(J)$ are assigned the same schedule if $\Lambda^T(I - J) = 0$. Equivalently, two variables $x_i(I)$ and $x_i(J)$ are assigned the same schedule if I and J lie on the same hyperplane defined by the normal vector Λ. Therefore, for such RIAs one can draw isotemporal hyperplanes (see Fig. 2) in the index space, and because of this geometric interpretation, these schedules are often referred to as hyperplanar schedules. We should note here that Rao *et al.* [26], [27] showed that algorithms implementable on systolic arrays are exactly those RIAs that admit uniform affine schedules.

Hyperplanar schedules, however, do not exist for all RIAs; one can show (see *e.g.* , [26], [29]) that the RIA in example 2 cannot have a hyperplanar schedule. Nonetheless, it turns out that the scheduling properties of an RIA can be determined from the information gathered during the execution of a so-called *computability analysis* procedure. The aim of the computability analysis is to determine whether the given RIA is *well-defined.* A well-defined RIA should not have any directed cycles (*i.e.* , the execution of a computation should not depend on its output) or any directed paths coming from infinity (*i.e.* , every computation can be completed in a finite number of steps) in its dependence graph. The computability analysis leads to an iterative decomposition of the RDG, which results in a tree structure called the *computability tree.* It is shown (see [33], [13], [26]) that an RIA cannot admit hyperplanar schedules if the depth of its computability tree, l, is greater than 1.

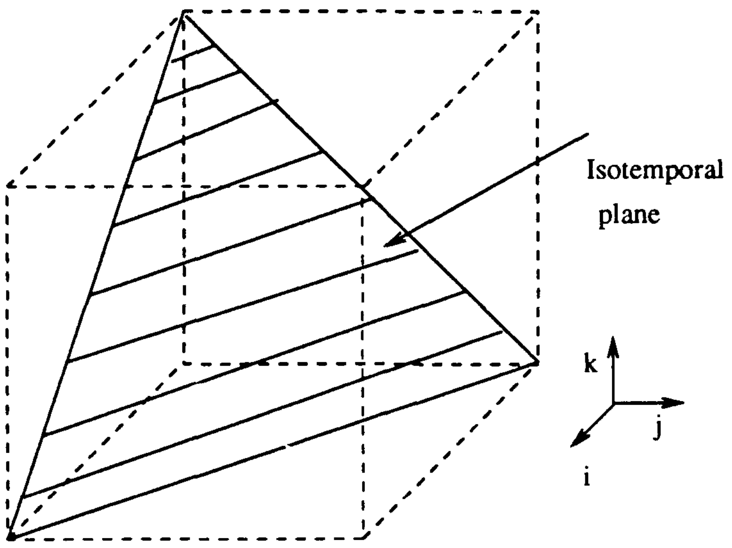


Figure 2: Isotemporal hyperplanes in the index space of an RIA that admits uniform affine schedules.

In our work, [37, 32] we have shown that hyperplanar schedules, valid for only a restricted subclass of RIAs, can be extended and that asymptotically optimal *subspace schedules* can be determined to schedule any RIA defined over a bounded or a semi-infinite index space. In particular, we show that if the depth of the computability tree is l, and the dimension of the index space is S, then for every indexed variable x_i, one can determine a linear subspace L_i of dimension $S - l$, such that two variables $x_i(I)$ and $x_i(J)$ can be scheduled at the same step if $(I - J) \in L_i$. Thus, the proposed scheduling scheme defines isotemporal surfaces that are linear subspaces of dimension $S - l$, instead of dimension $S - 1$ (which is the case for hyperplanar schedules). Fig. 3 shows the isotemporal lines in the index space of the RIA for 2-D filtering presented in example 2. We should mention here that we have also devised alternative algebraic techniques for scheduling RIAs defined over bounded index spaces. These algebraic scheduling techniques, however, do not always have the same geometrical interpretation as the subspace scheduling schemes and are not valid for RIAs defined over semi-infinite index spaces (note that, Karp *et al.* mostly studied RIAs defined over semi-infinite index spaces only.)

We also show that the subspace scheduling scheme described above, can be used to characterize the extent of parallelism in RIAs (*i.e.* , whether the given RIA has unbounded parallelism or not). This was attempted by Karp *et al.* [13] in their seminal paper; however, the general result that we prove here was left as a conjecture. In particular, we show that if the computability tree is of depth

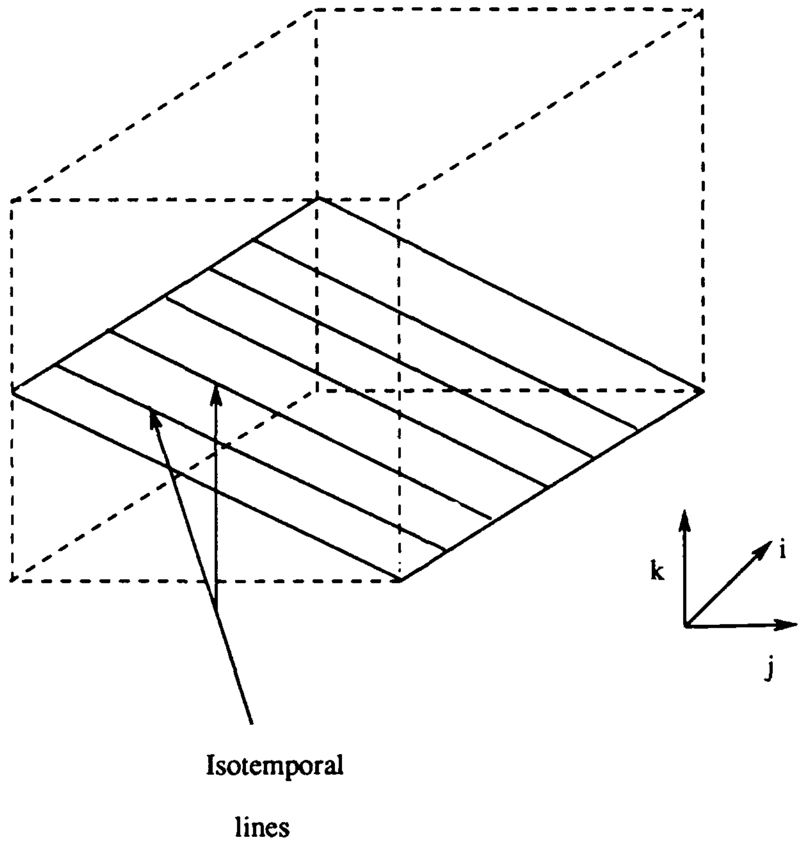


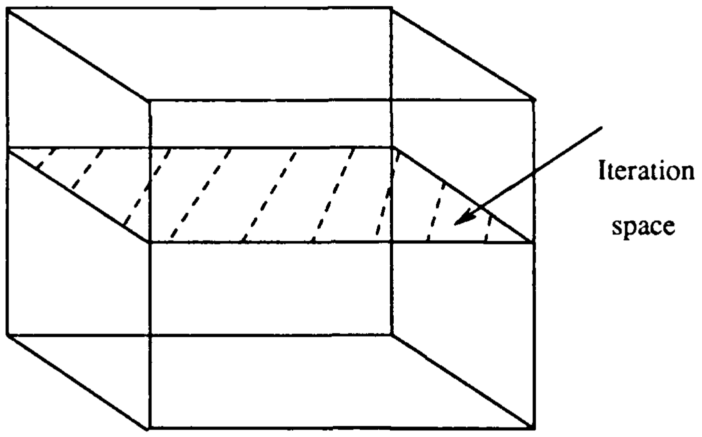
Figure 3: Isotemporal lines for variables y and w in the index space of the RIA in Example 2.1.

$l < S$ (where S is the dimension of the index space) then it always has unbounded parallelism, and if $l = S$ then it has bounded parallelism except possibly at the boundaries of the index space.

The next issue can be motivated by observing that for RIAs admitting hyperplanar schedules, one can write the schedule of a variable $x_i(I)$ explicitly as $\Lambda^T I + \gamma_{x_i}$. Rao *et al.* [26], [29], [27], [10], extended this result and obtained explicit schedules for all RIAs defined over bounded index spaces; however, explicit schedules for RIAs defined over semi-infinite index spaces have not been developed. The explicit scheduling functions are still affine, *i.e.* , the schedule for a variable $x_i(I)$ can be written as $\Lambda_i^T I + \gamma_{x_i}$; however, the vector Λ_i may be different for different variables and its entries along with γ_{x_i} will be functions of the size parameters (*i.e.* , the bounds) of the RIA. For RIAs that are defined over semi-infinite index spaces we show that we can identify a precisely defined subclass, such that the explicit schedule of a variable $x_i(I)$ can be expressed as a polynomial in the indices of the index point I. For RIAs not belonging to this subclass, we show that one can always come up with examples such that the schedule of $x_i(I)$ is exponential or double exponential in the indices of I. However, we are able to refine our analysis further and provide a necessary and sufficient condition for the schedule of a variable $x_i(I)$ to be polynomially bounded.

The results presented so far, on the analysis of parallelism in RIAs, are independent of implementation in any particular machine and can be used for executing RIAs in parallel on any architecture. Previous work by several researchers beginning with [21], however, has shown that RIAs are particularly well suited for implementation on regular mesh-connected processor arrays. The proposed technique is to define a linear subspace, called the *iteration space*, such that computations corresponding to variables $x_i(I)$ and $x_i(J)$ are assigned to the same processor if the vector, $I - J$ belongs to the iteration space. A geometric interpretation of the mapping technique can be obtained by decomposing the index space into parallel subspaces such that variables belonging to the same subspace are assigned to the same processor. Fig. 4 shows the iteration space and a resultant processor array for implementing the RIA in example 2. It can be shown that, because of the regular structure of dependence graphs of RIAs, resultant processor arrays are regular with fixed and local interconnections. This linear projection scheme for partitioning computations is only the first step towards parallel implementation, and one has to make sure that the partitioning scheme is *compatible* with scheduling techniques, *i.e.* , two computations scheduled at the same step should not be assigned to the same processor.

We have shown [26], [33], [29] that given an RIA that is scheduled by a subspace scheduling scheme as described before, one can always determine compatible iteration spaces. Recall that in our subspace scheduling, $x_i(I)$ and $x_i(J)$ are assigned the same schedule if $I - J$ belongs to an isotemporal subspace; note that, isotemporal subspaces may be different for different variables x_i. Thus, for an iteration space to be compatible with a subspace schedule, one should satisfy the following condition: if $I - J$ $(I \neq J)$ is in the iteration space then it should not be in any of the



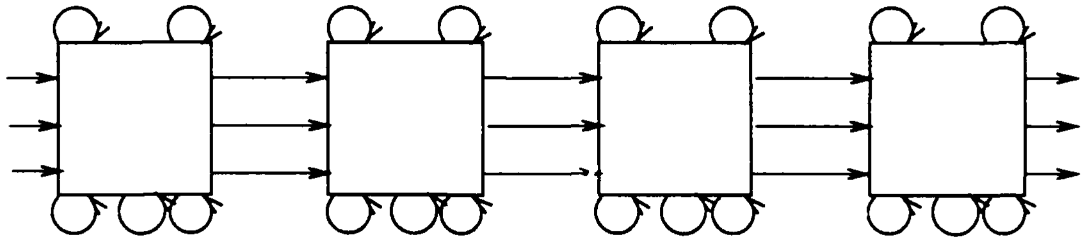


Figure 4: An iteration space and a corresponding processor array for the RIA in Example 1.2.

isotemporal subspaces and vice-versa. In other words, an iteration space will be compatible with a subspace schedule if and only if it has only zero intersection with the isotemporal subspace of each variable x_i in the RIA. Therefore, the procedure for designing processor arrays can be reduced to the linear algebraic problem of determining a subspace that has no non-zero intersection with a finite number of given subspaces.

We can now briefly summarize our new results on the analysis and implementation of RIAs as follows (these results are based on the novel subspace scheduling scheme that we have developed):

1. We have developed a scheme to schedule any given RIA defined over a bounded or semi-infinite index space. This generalizes the so-called hyperplanar schedules.

2. We have developed procedures to determine explicit schedules *i.e.* , formulas for the schedules for a variable $x_i(I)$ are expressed as a function of i, I and possibly also of the size parameters of the RIA. We have also analyzed the extent of parallelism in RIAs; this analysis leads to a proof of a conjecture made in the seminal paper of Karp *et al.* [13].

3. We have shown that for any given RIA scheduled according to our subspace scheduling scheme, there always exist compatible mesh-connected processor arrays for implementing it in parallel.

2.3 Systematic Formulation of RIAs

Let us first introduce the concept of *localized algorithms* that are close to RIAs (see *e.g.* , [37, 36, 35, 12, 33]). The definition of the localized algorithms is motivated by the observation that there are certain problems that can be solved by algorithms that have *regular* dependence graphs that are not completely homogeneous. That is, the dependence graphs may have dependencies or computations that are present only in certain portions of the dependence graphs. As we shall discuss in [37], one way of handling such cases is to assume that the dependences and the computations are present everywhere in the index space and then to apply the results for RIAs. There are several problems where this approach is reasonable; for example the Gaussian elimination algorithm without pivoting can be first written in the localized algorithm form and then can be implemented on processor arrays by modeling the localized algorithm as an RIA. The other approach is to break up the dependence graph into more than one component such that the dependence graph is homogeneous over each component. The mapping techniques can then be applied to each such component with special consideration to the dependences at the boundaries between the components. The latter approach is discussed in more detail in [37] where the example of Gauss-Jordan elimination algorithm is worked out.

The localized algorithms have statements of the form

$$x_i(I) = f_i(x_1(I - d_{i1}), \cdots, x_V(I - d_{iV})) \quad \forall\, I \in \mathbf{I}_i. \tag{1}$$

Thus each statement in this algorithm may have a different index space of its own; as a comparison, all statements in an RIA have the same index space.

Partial attempts have been made by several authors, including [14], [22], [15], [25], [4] and [7], to formalize the conversion procedure for going from an initial representation to an RIA or a localized algorithm. The first step always is to convert algorithms into equivalent Single Assignment Codes (SACs) and the second step tries to localize the dependences by eliminating broadcasts. Single assignment codes [2] are representations where every variable defined in the algorithm takes on a unique value during the course of execution. The fact that the dependence graph of an algorithm can be easily determined from its SAC, has made SACs a very useful starting representation for parallel implementations of algorithms. Considering its importance, a lot of work has been done in trying to convert sequential algorithms into SACs, see *e.g.,* [24]. However, sequential algorithms are not the only representations from which SACs can be derived. Often SACs can be derived systematically from given mathematical expressions. Consider a mathematical expression for matrix multiplication

$$\textbf{For all} \quad \text{tuples } (i,j)\ ,\ 1 \le i,j \le n \ \textbf{do}$$

$$c_{ij} := \sum_{\text{for all } 1\le k\le n} a_{ik}\cdot b_{kj}. \tag{2}$$

and a SAC

$$\textbf{For all} \quad \text{triples } (i,j,k),\quad 1 \le i,j,k \le n \ \textbf{do}$$

$$c(i,j,k+1) := c(i,j,k) + a_{ik}\cdot b_{kj} \tag{3}$$

In the mathematical expression, the ordering of operations in the inner product is not specified and in fact it can be arbitrary because of the commutativity and associativity of the operation +. However, in the given SAC the ordering is fixed and a degree of freedom has been lost. Since the original representation has more freedom and potential parallelism in it, it would be desirable to make it the starting representation and then systematically derive one or more SACs from it. It turns out that a number of algorithms can be written in the form of (2) (see [37, 36, 35]), and we shall refer to such representations as *Assignment Codes* (ACs). The prefix *'Single'* has been intentionally dropped to emphasize the fact that in such representations the number of inputs for computing a variable may depend on the problem size, as opposed to a conventional SAC where the number of inputs to every variable is restricted to be some constant, independent of the problem size. From now on we shall refer to the number of inputs to a variable as its in-degree and the number of variables that a particular variable is input to as its out-degree. If the in- or out-degree of a variable depends on the problem size, then we shall define it to be unbounded. Thus, the variables in ACs can have unbounded in- and out-degrees, whereas in SACs the variables have bounded (*i.e., constant*) in-degrees but may have unbounded out-degrees.

We shall further restrict ourselves to *linearly indexed* ACs, which can be shown to be very close to mathematical expressions for a number of problems, especially in signal processing and matrix algebra. A linearly indexed AC has statements of the form

$$x(PI + \mathbf{d}) \text{ depends on } y(QI + \mathbf{e}) \text{ for all } I \in \mathbf{I} \subset \mathbf{Z}^S \tag{4}$$

where P and Q are integral matrices independent of I, $\mathbf{I}$ is an index space which is the set of all lattice points enclosed within a specified region in a S-dimensional Euclidean space and $\mathbf{d}$. $\mathbf{e}$ are constant displacement vectors. P and Q are often referred to as the *indexing matrices*. We have shown in [37, 36, 35] that in- and out-degrees of variables x and y are completely determined by the structure and dimension of the right null-space of each of the indexing matrices. Many algorithms are actually directly available as (4), and examples include the formulas for matrix multiplication, any m-dimensional convolution/correlation, matrix transposition, and solving matrix Lyapunov's equation. Algorithms that are not directly in the form of (4) can often be easily put in that form by analyzing their sequential representations (see [37]).

Example 3.1: The formula for matrix multiplication is:

$$\textbf{For all} \quad \text{tuples } (i,j)\ ,\ 1 \le i,j \le n \ \textbf{do}$$

$$c_{ij} := \sum_{\text{for all } 1 \le k \le n} a_{ik} \cdot b_{kj}$$

The index space of the example is $\mathbf{I} = \{(i,j,k) \mid 1 \le i,j,k \le n\}$. There is one functional relation in the given AC with the dependence matrices

$$P_c = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix} \qquad Q_{ac} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix} \qquad Q_{bc} = \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

and the displacement vectors $\mathbf{d}, \mathbf{e} = 0$. $\square$

We have shown that a linearly indexed AC can be systematically decomposed into a linearly indexed SAC and a linearly indexed *dual SAC*. Linearly indexed dual SACs may have variables with unbounded in-degrees but bounded out-degrees, as opposed to the linearly indexed SACs, which have variables with bounded in degrees but possibly unbounded out degrees. Formal procedures will be then outlined for converting linearly indexed SACs and linearly indexed dual SACs into localized algorithms. The conversion of linearly indexed SACs to localized algorithms involves eliminating global dependencies by propagating variables in a systematic manner in the index space. The conversion of dual SACs to localized algorithms is achieved by distributing computations and introducing an ordering among the computations. The two conversion procedures turn out to be duals of each other. We should mention here that starting with linearly indexed ACs is by no means essential in our approach; if one cannot find a AC easily, then one can try to use other well-known techniques and start the procedure with a linearly indexed SAC.

In summary, we have developed a hierarchical procedure for going from a higher level representation of an algorithm to a localized algorithm, which can be described by an RIA or a localized algorithm. It can be described as follows:

Mathematical Description $\rightarrow$ Linearly Indexed Assignment Codes $\rightarrow$ Linearly Indexed Single Assignment Codes and dual Single Assignment Codes $\rightarrow$ Localized Algorithms.

The conversion procedure is by no means unique and a number of localized algorithms can be generated starting from the same AC. To enable an efficient choice we have also developed procedures to directly schedule and analyze linearly indexed codes of the form (4). For example, we have developed necessary and sufficient conditions for determining whether a sequence of SACs of the form (4) can be scheduled using affine schedules (see [37, 12, 33]). Procedures to schedule linearly indexed codes that do not admit affine schedules are also discussed in [37].

3 Summary of Our Work on Reconfigurable Arrays

As evident from our work on regular iterative arrays (summarized in Section 2), an array of identical processing elements is an indispensable architecture in the VLSI and WSI technology and proves itself very useful in parallel processing applications. Often, however, during the fabrication process or during run-time, some of the processing elements in a large array are inevitably going to be faulty. Spare PEs and extra routing hardware are often provided so that a fault-free array can be constructed; such reconfiguration capability can be used to increase the yield, and to guarantee fault tolerance in applications when failure is not permissible. Our work in this regard has been concerned with the design and analysis of such configurable fault-tolerant arrays.

The general model considered by us is shown in Fig. 5 (see *e.g.* , [20, 23, 30, 39, 40, 41, 42]); it consists of a set of identical processors embedded in a flexible interconnection structure that is configured in the form of a rectangular grid. Each grid line in the mesh has a fixed number of data paths that can be routed along it (*i.e.* , the model has fixed channel width); switches can be placed at every grid point and at every location where a processor is connected to the grid. Furthermore, often the processors are divided into a set of non-spare PEs (say an $m \times n$ array) and a set of spare PEs that are distributed in a pre-determined fashion.

Given a set of faulty PEs, the objective is to reconfigure the connections among the PEs such that a new rectangular *logical array* is formed comprising only the healthy PEs and demanding no more hardware resources (*e.g.*, spare PEs, tracks, and switches) than available. It is obvious that the more the additional hardware, the higher is the reconfiguration probability. Nevertheless, space and cost limitations might make it impossible to add as much hardware as one would want. Such considerations lead to the following design question: *What is the optimal amount of required hardware resources, i.e., the number of spare PEs, the channel width and the distribution of routing* *switches, such that the resulting architecture has high reconfigurability?* A related important question addresses the ease of reconfiguration: *Given a configurable architecture with fixed resources, are there efficient and simple algorithms for reconfiguring such architectures with high probability?*

In our recent work we have provided partial answers to both the above issues. However, to further motivate our results and contributions, and to introduce some of the relevant concepts we shall first briefly review the previous work in this area.

3.1 Background

In the context of reconfiguration of processor arrays, several researchers [8, 18] have addressed important analytical issues of the following nature. Given an $n \times n$ array of PEs (or cells), each of which can be faulty with probability p: (1) Can one devise 'good' algorithms that will connect the non-faulty cells in the form of a grid *with high probability*? (2) With high probability, what are the required channel width (defined as the maximum number of interconnection paths routed along any grid line) and the length of the longest interconnecting wire?

As mentioned before, instead of asymptotic analysis, our studies have been motivated by a more practical query: how to reconfigure arrays with only a small amount of extra routing hardware. A general methodology to reconfigure arrays with faulty PEs is to determine the so-called *compensation paths.* A compensation path is comprised of a sequence of substitutions that logically replaces a faulty PE by a spare one, and can be described as follows. Let a non-spare PE at location (x, y) be faulty, then in any valid reconfiguration it has to be replaced by a healthy processor. Let the faulty PE at (x, y) be *logically replaced* by a healthy PE, say at location (x', y'); logical replacement implies that in the reconfigured array the physical PE at location (x', y') will be reindexed as (x, y). The PE at (x', y') is in turn replaced by a healthy PE, say at location (x'', y''); one can continue this chain until one ends up at a spare PE. Now a compensation path can be defined as the ordered sequence of nodes (x, y), (x', y'), (x'', y''), $\cdots$, involved in the replacement chain. Thus, the compensation paths determine neighbors of each PE in the *logical* or the reconfigured array. Hence, once the compensation paths are determined, the reconfiguration procedure is completed by connecting each PE to its logical neighbors.

It is easy to see that if the number of faulty PEs is less than the number of spare PEs, then one can always define a set of compensation paths for successful reconfiguration. However, the characteristics of the compensation paths (*e.g.*, the geometrical distances between consecutive nodes, or the relative positions of the nodes in the grid) determine the amount of routing hardware needed to implement the necessary connections among the logical neighbors. It can be easily shown that if the number of routing tracks is fixed, then one cannot allow arbitrary sets of compensation paths. In other words, by limiting the hardware resources one limits the number of faulty patterns that one can reconfigure. *Hence, a natural question to ask is how many tracks should one provide so as* *to allow a large enough class of compensation paths, and yet keep the hardware redundancy low.*

A model with very limited hardware resources has been studied in [16, 17, 38]. It consists of an $m \times n$ array of non-spare PEs, 1 row (or column) of spare PEs along each boundary, a single-track along every grid line (*i.e.*, channel width = 1), and single-track switches located at intersections where processors are connected to the grid. It is further assumed that a faulty PE can be converted into a connecting element, thereby making an implicit assumption that there is an extra channel within every PE (because of the assumption such models have also been referred to as an $1\frac{1}{2}$-track model). The single-track switch model's advantages arises from its inherent simplicity: since data paths take up significant amount of area on a wafer/chip, considerable saving in area is achieved by allowing only one data path along every grid line; moreover, the simplicity of the switches makes the routing hardware more reliable. Furthermore, extensive simulations reported in [17] show that considerable enhancement in yield can be achieved by reconfiguring the array grid models with single-track switches.

We now briefly discuss the results reported in [17]. The paper derives a set of sufficient conditions (stated in the form of a so-called *reconfigurability theorem* presented below) for determining whether an array with a particular distribution of faulty processors is *reconfigurable*; where, a given array is *reconfigurable* if the non-faulty processors can be connected to form an $m \times n$ array. The sufficient conditions restrict the compensation paths to be straight. Fig. 7 shows a compensation path and the corresponding routing required for replacing a single faulty processor in the single-track model; note that the compensation path is straight and continuous. This simple concept of using straight and continuous compensation paths can be also used in the presence of multiple faulty processors and the sufficient conditions can be formally presented in the form of the so-called reconfigurability theorem (for a formal proof, see [16, 17]):

Reconfigurability Theorem: *Given an $m \times n$ array of non-spare PEs, with spare PEs along the sides, it is reconfigurable into an $m \times n$ array of healthy processors by single-track switches if 1) there exists a set of continuous and straight compensation paths covering all the faulty non-spare PEs and 2) there is neither intersection or near-miss among the compensation paths.*

A near-miss situation occurs if two compensation paths in neighboring rows (columns) overlap and are in opposite directions (see Fig. 8; note that a near-miss situation does not occur if the compensation paths overlap by only one node).

In [17] an algorithm to determine valid reconfigurations that satisfy the conditions in the reconfigurability theorem is also presented. The algorithm is developed by reformulating the reconfigurability problem as a maximum independent set problem, and then adapting a well known algorithm for determining maximum independent sets in a graph. However, the maximum independent set problem is NP-complete and the best known algorithms take exponential time; hence, the algorithm presented in [17] has exponential complexity. *The question whether efficient polynomial time* *algorithms exist was left as an open one. Moreover, efficient algorithms were not known even for the restricted cases where spare processors are available only along, for example, two or three sides (as opposed to on all four sides as shown in Fig. 6).*

3.2 New Contributions

In view of the above results, the contributions of our work with regard to the single-track models can be summarized as follows (see [38]):

- We showed that the conditions in the reconfigurability theorem are *not necessary* correcting a claim made in [11].
- We developed a polynomial time algorithm (in fact, the complexity is $O(|F|^2)$, where $|F|$ is the number of faulty processors) for determining valid reconfigurations according to the sufficient conditions. Moreover, linear time algorithms for determining valid reconfigurations are developed for the restricted cases where the spare processors are not present along all four sides of the array.

We should note here that the combinatorial problem underlying the reconfigurability issues in the single-track model is a very interesting geometrical problem on rectangular grids, and can be stated as follows:

Problem 1 *Let V be the set of grid points in an $m \times n$ 2-dimensional grid, and let $F \subset V$. Determine a set of straight lines such that*

1. *Each vertex $v \in F$ is assigned a straight line connecting it to one of the four boundaries of the grid.*
2. *The straight lines are non-intersecting.*

The algorithm developed by us appears to be the first-known polynomial time algorithm for this problem.

The algorithms discussed so far focus on the *satisfiability* question, *i.e.*, whether *all the faulty PEs* can be replaced by straight and non-intersecting compensation paths. Often, however, a more relevant issue might be to determine the maximum number of faulty processors that can be replaced. In [1] a polynomial time algorithm of time complexity $O(|F|^3)$ was developed for solving the corresponding combinatorial problem.

The motivation behind some of the results to be described next is to introduce *a minimal amount of additional hardware, that will allow a much larger class of compensation paths than the restricted class of straight paths.* One can easily observe that the sufficient conditions for reconfiguration as discussed above are also valid for more general array models such as the ones with multiple-tracks. One, however, hopes that for such more powerful models it should be possible to develop more general conditions to allow reconfiguration of arrays that otherwise could not be reconfigured in the single-track model. With such motivation in mind, we first considered an augmented single-track switch model as shown in Fig. 9. We showed that the augmented model is more powerful than the simple single-track model: the compensation paths in the augmented model need not be straight any more and can have *bends.* In general, if one allows multiple data paths along every grid line (*i.e.* , a multiple-track model), then the compensation paths can be crooked and the restriction of near misses is no longer required. *Hence, a generalized sufficient condition can be stated as follows: an array grid model with multiple-track switches is reconfigurable if one can determine a set of non-intersecting compensation paths (continuous, but not necessarily straight) for the faulty PEs in the array.* We also showed that the combinatorial problem corresponding to such a sufficient condition can be efficiently solved by reducing it to the well known problem of determining maximum-flow in networks.

In a more recent work, we have made progress in developing new reconfiguration algorithms (see [44]) for a different set of models that were first introduced in [39] (see also [3, 20, 46]). The appeal of this set of models is its hardware simplicity. It consists of an $N \times (N+1)$ array, where N spare PEs are configured in the form of a spare column, and are located along one boundary of the original $N \times N$ array. The goal of the reconfiguration algorithm is to obtain an $N \times N$ array of healthy processors given that any N of the PEs are faulty; moreover, the reconfigured array must satisfy certain *neighborhood* constraints. Let the *physical array* be the array given to us, and let the *logical array* be the reconfigured array. Then the neighborhood constraint requires that in the logical array the neighbors of a healthy processor be restricted to lie in a fixed neighborhood around the healthy processor.

The reconfiguration algorithms presented in [39] for the above model are very fragile, and fail to reconfigure even when the faulty PE distributions are very simple; the following are only two simple instances (*one can generate several other instances*): (1) If the bottom row or the top row does not have any faulty processors, (2) If there are two columns each with a stack of faulty processors, even of size two.

The algorithms that we have developed can reconfigure *all instances of arrays that the algorithms reported in [39] can.* Moreover, we can also reconfigure arrays with several other faulty distributions, *without relaxing the neighborhood constraints.* In particular, we give reconfiguration algorithms for the following cases (1) If every column has one faulty PE (2) If one column has two faulty PEs and the rest of the columns have one faulty PE each or no faulty PEs, (3) If there are two stacks of faulty PEs, each of length $N/2$, and (4) If the faulty PEs can be partitioned into blocks such that each block can be considered as one of the above special cases.

The reconfiguration algorithms are based on one simple algorithm that shows how to reconfigure an $N \times (N+1)$ array into an $(N+1) \times N$ array and vice versa. Moreover, our algorithms generate very regular patterns, and can be implemented in a distributed fashion instead of being processed by one host processor. We should also mention here that in [3], a model similar to ours is considered; however, they do not impose the strict neighborhood restrictions. *We can show that the algorithms in [3] will require larger neighborhood for arrays that can be reconfigured with smaller neighborhoods with our algorithm;* the details will be provided in the full paper.

We are continuing our studies on developing more powerful models that have high reconfiguration probability and yet does not require too much extra hardware.

References

[1] J. Bruck and Vwani Roychowdhury. How to Play Bowling In Parallel on the Grid? Technical report, Tech. Rep. RJ 7209 (67688), IBM, Almaden Research Center, San Jose, CA., Dec. 1989.

[2] S. J. Celoni and J. L. Hennessy. SAL-A Single Assignment Language for Parallel Algorithms. Technical report, Computer Systems Laboratory, Dept. of Electrical Eng., Stanford University, Stanford, California, July 1981.

[3] M. Chean and J. A. B. Fortes. The full-use-of-suitable-spares (fuss) approach to hardware reconfiguration for fault-tolerant processor arrays. *IEEE Transactions on Computers*, 39:564-571, April 1990.

[4] J. M. Delosme and I. C. F. Ipsen. An Illustration of a Methodolgy for the Construction of efficient Systolic Architectures in VLSI. In *Proceedings Second Int. Symp. on VLSI Tech.* Taipei, Taiwan, 1985.

[5] E. Deprettre and P. Dewilde. Orthogonal cascade realization of real multi-port digital filters. Technical report, Network Theory Section, Delft Univ., The Netherlands, 1980.

[6] A. Fettweis. Digital Filter Structures Related to classical Filter Networks. *Arch. Eleck. Ubertragung*, 25:79-89, Feb. 1971.

[7] J. A. B. Fortes and D. I. Moldovan. Data broadcasting in linearly scheduled array processors. *Proc. 11th Ann. Symp. on Computer Architecture*, pages 224-231, June 1984.

[8] J. W. Greene and A. El Gamal. Configuration of VLSI arrays in the presence of defects. *JACM*, 31, No. 4:694-717, October 1984.

[9] H. V. Jagadish. *Techniques for the Design of Parallel and Pipelined VLSI Systems for Numerical Computations.* PhD thesis, Stanford University, Stanford, California, Dec. 1985.

[10] H. V. Jagadish, S. K. Rao, and T. Kailath. Multi-processor architectures for iterative algorithms. *Proceedings of the IEEE*, 75, No. 9:1304–1321, Sept. 1987.

[11] S. N. Jean and S. Y. Kung. Necessary and sufficient conditions for reconfigurability in single-track switch WSI arrays. in Proc. of Int. Conf. on Wafer Scale Integration, January 1989.

[12] T. Kailath and V. Roychowdhury. Scheduling Linearly Indexed Assignment Codes. In *Proc. SPIE Symposium on High Speed Computing II, vol: 1058, Los Angeles, CA*, pages 118–129, January 1989.

[13] R. M. Karp, R. E. Miller, and S. Winograd. The organization of computations for uniform recurrence equations. *Journal of the ACM*, 14:563–590, 1967.

[14] R. M. King. Research on the synthesis of concurrent computing systems. *Proc. of the 10th Symp. on Computer Architecture*, pages 39–46, 1983.

[15] R. H. Kuhn. Transforming algorithms for single-stage and VLSI architectures. *Proc. Workshop on Interconnection Networks for Parallel and Distributed Processing*, pages 11–19, Apr. 1980.

[16] S. Y. Kung. *VLSI Array Processors.* Prentice Hall, 1987.

[17] S. Y. Kung, S. N. Jean, and C. W. Chang. Fault-tolerant array processors using single-track switches. *IEEE Trans. on Computers*, 38, No. 4:501–514, April 1989.

[18] T. Leighton and C. E. Leiserson. Wafer-Scale Integration of Systolic arrays. *IEEE Trans. on Computers*, C-34, No. 5:448–461, May 1985.

[19] C. E. Leiserson, F. M. Rose, and J. B. Saxe. Optimizing synchronous circuitry by retiming. *Proceedings of the CalTech Conference on VLSI*, pages 87–116, March 1983.

[20] F. Lombardi, M. G. Sami, and R. Stefanelli. Reconfiguration of VLSI arrays by covering. to appear in IEEE Trans. on CAD, 1989.

[21] D. I. Moldovan. On the analysis and synthesis of VLSI algorithms. *IEEE Trans. on Computers*, C-31:1121–1126, Nov. 1982.

[22] D. I. Moldovan. On the design of algorithms for VLSI systolic arrays. *Proceedings of the IEEE*, pages 113–120, Jan. 1983.

[23] W. R. Moore. A review of Fault-tolerant Techniques for the enhancement of Integrated Circuit yield. *Proc. of the IEEE*, pages 684–698, May 1986.

[24] Y. Muraoka. *Parallelism Exposure and Exploitation in Programs.* PhD thesis, University of Illinois, Urbana-Champaign, Feb. 1971.

[25] H. Nelis, E. F. Deprettere, and J. Bu. Automatic design and partitioning of VLSI systolic/wavefront arrays. In *Proc. SPIE Conference 1987, San Diego*, 1987.

[26] S. K. Rao. *Regular Iterative Algorithms and their Implementation on Processor Arrays.* PhD thesis, Stanford University, Stanford, California, 1985.

[27] S. K. Rao, V. P. Roychowdhury, and T. Kailath. *Systolic Arrays and their Extensions.* Prentice Hall, to appear in 1991.

[28] S. K. Rao and T. Kailath. What is a Systolic Algorithm? In *SPIE Proceedings.* Real-Time Signal Processing, 1986.

[29] S. K. Rao and T. Kailath. Regular Iterative Algorithms and their Implementations on Processor Arrays. *Proceedings of the IEEE*, 6, no.3:259–282, March 1988.

[30] A. L. Rosenberg. The Diogenes approach to testable fault-tolerant array of processors. *IEEE Trans. on Computers*, pages 902–910, Oct. 1983.

[31] V. P. Roychowdhury and T. Kailath. Regular Processor Arrays for Matrix Algorithms with Pivoting. *Int. Conf. on Systolic Arrays*, San Diego, CA,:237–246, May 1988.

[32] V. P. Roychowdhury and T. Kailath. Subspace Scheduling and Parallel Implementation of Non-systolic Regular Iterative Algorithms. *Journal of VLSI Signal Processing*, vol.1:127–142, December, 1989.

[33] V. P. Roychowdhury and T. Kailath. Scheduling linearly indexed assignment codes. Submitted to Journal of Parallel and Distributed Computing, Jan. 1989.

[34] V. P. Roychowdhury and T. Kailath. Regular Processor Arrays for Matrix Algorithms with Pivoting. Submitted to CACM, March, 1988.

[35] V. P. Roychowdhury, L. Thiele, S. K. Rao, and T. Kailath. On the Localization of Algorithms for VLSI Processor Arrays. *IEEE Workshop on VLSI Signal Processing*, Monterey, CA,:237–246, Nov. 1988.

[36] V. P. Roychowdhury, L. Thiele, S. K. Rao, and T. Kailath. On the Localization of Algorithms for VLSI Processor Arrays. Submitted to IEEE Trans. on Computers, September, 1988.

[37] Vwani P. Roychowdhury. *Derivation, Extensions, and Parallel Implementations of Regular Iterative Algorithms.* PhD thesis, Stanford University, Stanford, California, Dec. 1988.

[38] V. P. Roychowdhury, J. Bruck, and T. Kailath. Efficient Algorithms for Reconfiguration in VLSI/WSI Arrays. *IEEE Trans. on Computers: Special Issue on Fault tolerant Computing*, 39, No. 4:480–489, April 1990.

[39] M. Sami and R. Stefanelli. Reconfigurable architectures for VLSI processing arrays. *Proc. of the IEEE*, pages 712–722, May 1986.

[40] D. P. Siewiorek and R. S. Swarz. *The Theory and Practice of Reliable System Design*. Digital Press, 1982.

[41] L. Snyder. Introduction to the configurable, highly parallel computer. *IEEE Trans. on Computers*, 15:47–56, Jan. 1982.

[42] S. K. Tewksbury. *Wafer-Level Integrated Systems: Implementation Issues*. Kluwer Academic Publishers, 1989.

[43] P. P. Vaidyanathan and S. K. Mitra. A general theory and synthesis procedure for low sensitivity digital filters. ECE Report, Dept. of Electrical Eng., UCSB, California, Sept. 1982.

[44] T. Varvarigou, V. Roychowdhury, and T. Kailath. Some New Algorithms for Reconfiguring VLSI/WSI Arrays. In *Proc. International Conference on Wafer Scale Integration*, pages 229–235, January 1990.

[45] W. M. Waite. Path detection in multi-dimensional iterative arrays. *Journal of the ACM*, 14, 1967.

[46] H. Y. Youn and A. D. Singh. Bounded Channel Width Restructuring Algorithm for VLSI/WSI Arrays With Channel Faults. Presented at HFTM, June 1989, University of Illinois, Urbana, IL, USA., June 1989.

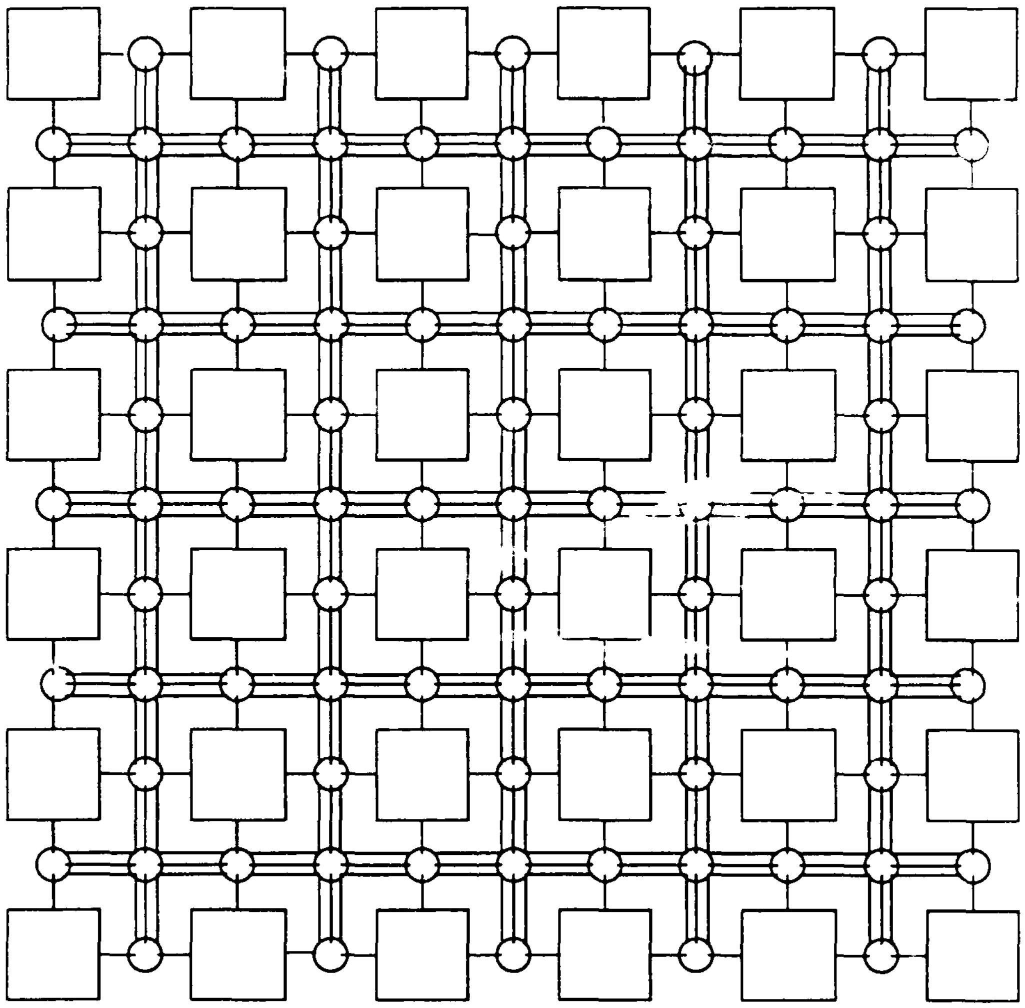
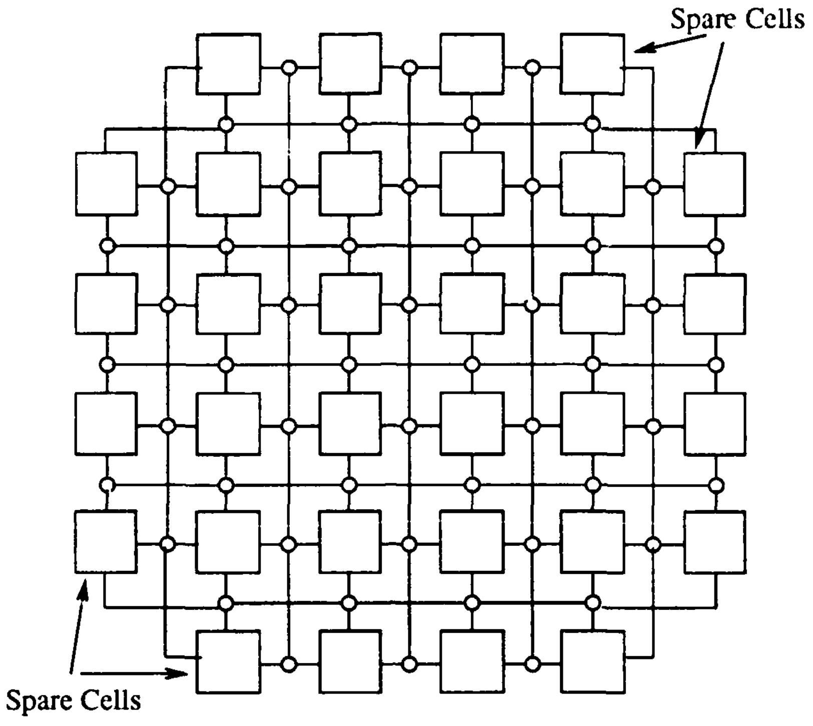


Figure 5: A general model for a reconfigurable processor array; the circles show possible locations of switches in the array.



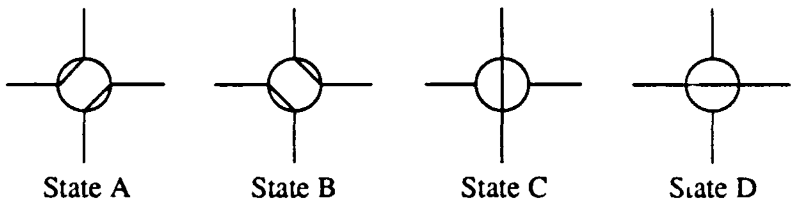


Figure 6: The array grid model based on single-track switches, shown with the possible states of a switch.

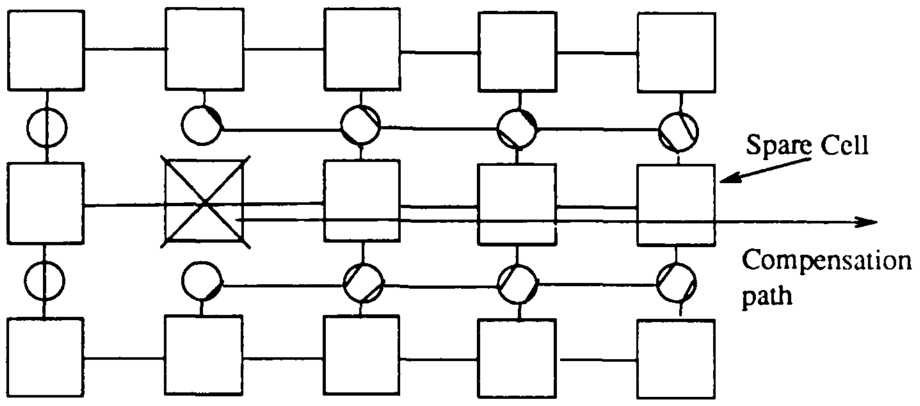


Figure 7: A compensation path and the corresponding routing required for replacing a single faulty processor in the single-track model.

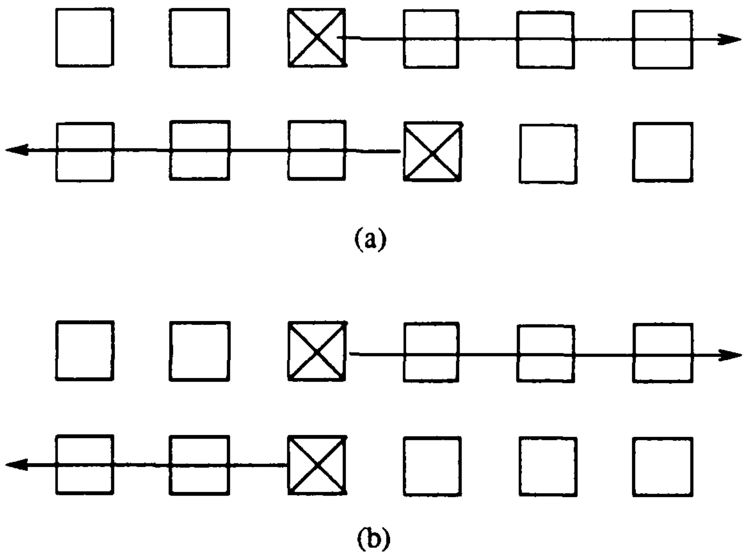
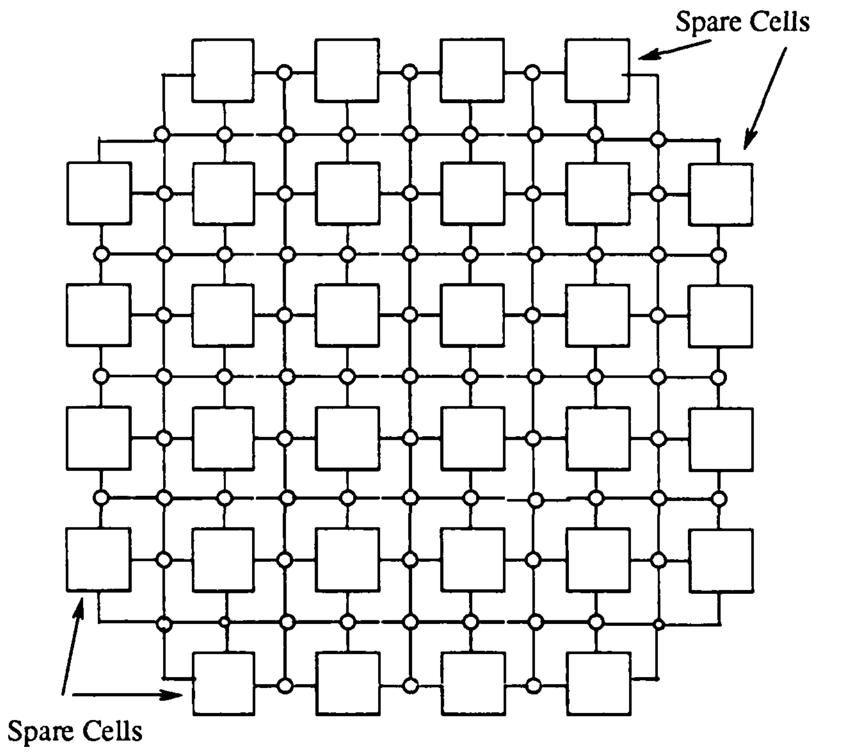


Figure 8: Near-miss situations among compensation paths: figure(a) shows a near-miss situation; figure(b) shows a situation that does not correspond to a near-miss.



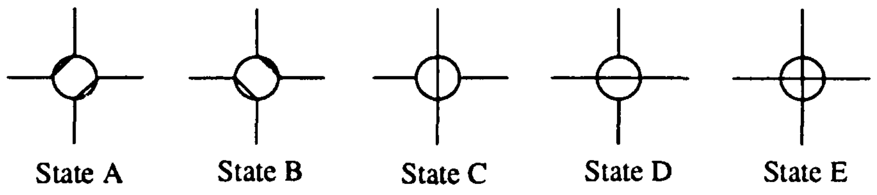


Figure 9: An augmented single-track model.

4 Publications List

Published Manuscripts

1. Y. Yoganadam, V. U. Reddy and T. Kailath, "Performance Analysis of the Adaptive Line Enhancer for Sinusoidal Signals in Broadband Noise," *IEEE Trans. ASSP*, Vol. 36, no. 11, pp. 1749-1757, November 1988.
2. A. Dembo, "Simple proof of the Concavity of the Entropy Power with Respect t Added Gaussian Noise" *IEEE Trans. on IT*, Vol. 35, no. 4, pp. 887-888, July 1989.
3. J. Chun and T. Kailath, "A Constructive Proof of the Gohberg-Semencul Formula," *Linear Algebra and Its Appls.*, Vol. 121, pp. 475-489, August 1989.
4. V. P. Roychowdhury and T. Kailath, "Subspace Scheduling and Parallel Implementation of Non-Systolic Regular Iterative Algorithms," *Journal of VLSI Signal Processing*, Vol. I, pp. 127-142, October 1989.
5. V. P. Roychowdhury, J. Bruck and T. Kailath, "Efficient Algorithms for Reconfiguration in VLSI/WSI Arrays," *IEEE Trans. Computers*, Vol. 39, no. 4, pp. 480-489, April 1990.

Accepted for Publication

1. J. Chun and T. Kailath, "Displacement Structure for Hankel, Vandermonde and Related (Derived) Matrices," *Linear Algebra and Its Appls.*
2. J. Chun and T. Kailath, "Divide-and-Computer Solutions of Least-Squares Problems for Matrices with Displacement Structure," *SIAM Journal of Matrix Analysis.*

Conference Papers

1. T. Kailath, "VLSI Array Processors for Communications, Control and Signal Processing," *TENCON 87*, pp. 96-98, Seoul, Korea, August 25-28, 1987.
2. P. G. Gulak, T. Kailath, A. Montalvo and V. P. Roychowdhury, "Decoding of Rate K/N Convolutional Codes in VLSI," *Dayton Conference on Systems Engineering*, Dayton, OH, September 1987.
3. D. Pal and T. Kailath, "Inertia Extraction of Bezoutians and Root Location of Polynomials Part I: Imaginary Axis Case," *Proc. 22nd Conference on Information Sciences and Systems*, pp. 964-969, March 1988.
4. V. P. Roychowdhury and T. Kailath, "Regular Processor Arrays for Matrix Algorithms with Pivoting," *Intern'l. Conference on Systolic Arrays*, pp. 237-246, San Diego, CA, May 1988.
5. J. Chun and T. Kailath, "Divide-and-Computer Solutions of Least-Squares Problems for Matrices With Displacement Structure," *Sixth Army Conference on Applied Mathematics and Computing*, Boulder, CO., May 31-June 3, 1988.

6. J. Chun and T. Kailath, ''Displacement Structure for Hankel- and Vandermonde-like Matrices,'' IMA Conference on Signal Processing, June 27-August 5, 1988, Minneapolis, MN. Reprinted, *Signal Processing and Mathematics*, ed. L. Auslander, T. Kailath and S. Mitter, Springer-Verlag, 1990.

7. V. P. Roychowdhury, P. G. Gulak, A. Montalvo, and T. Kailath, ''Decoding of Rate *k/n* Convolutional Codes in VLSI,'' *1987 Princeton Workshop on Algorithms, Architectures and Technology Issues for Models of Concurrent Computation*, N.J., September 1987. Reprinted in *Concurrent Computations: Algorithms. Architecture and Technology*, Chapter 33, eds. S.K. Tewksbury, B. W. Dickinson and S. C. Schwartz, Plenum Press, N.Y., July 1988.

8. J. Chun and T. Kailath, ''Generalized Displacement Structure for Block-Toeplitz, Toeplitz-Block, and Toeplitz-Derived Matrices,'' *NATO Conference on Linear Algebra, Signal Processing and Computer Architectures*, Leuven, Belgium, August 1988. Also, Springer-Verlag, eds. G. Golub and P. Van Dooren, 1989.

9. J. Chun, V. Roychowdhury and T. Kailath, ''Systolic Array for Solving Toeplitz Systems of Equations,'' *32th SPIE Symposium for Advanced Algorithms and Architectures for Signal Processing*, San Diego, CA, August 1988.

10. V. P. Roychowdhury, S. K. Rao, L. Thiele and T. Kailath, "On the Localization of Algorithms for VLSI Processor Arrays'', 1988 IEEE Workshop on VLSI Signal Processing, pp. 459-470, Monterey, CA, November 1988.

11. T. Kailath and V. P. Roychowdhury, ''Scheduling Linearly Indexed Assignment Code,'' *Proc. SPIE*, pp. 118-129, Los Angeles, CA, January 1989.

12. O. Farotimi, A. Dembo and T. Kailath, ''Absolute Stability and Optimal Training for Dynamic Neural Networks,'' *23rd Asilomar Conference on Circuits, Signals and Computers*, Asilomar, CA, November 1989.

13. A. Swindlehurst and T. Kailath, ''2-D Parameter Estimation Using Arrays With Multidimensional Invariance Structure,'' *23rd Asilomar Conference on Circuits, Signals and Computers*, Asilomar, CA, November 1989.

14. A. Swindlehurst, B. Ottersten and T. Kailath, ''An Analysis of MUSIC and Root-MUSIC in the Presence of Sensor Perturbations,'' *23rd Asilomar Conference on Circuits, Signals and Computers*, Asilomar, CA, November 1989.

15. D. Pal and T. Kailath, ''Fast Modified Triangular Factorization of Hankel, Quasi-Hankel and Sign-Modified Quasi-Hankel Matrices With Arbitrary Rank Profile,'' *23rd Asilomar Conference on Circuits, Signals and Computers*, Asilomar, CA, November 1989.

16. T. Kailath, J. Chun and V. Roychowdhury, ''Systolic Array for Solving Toeplitz Systems of Equations,'' *INDO-US Workshop*, Bombay, India, December 1989.

17. T. Varvarigou, V. P. Roychowdhury and T. Kailath, ''Some New Algorithms for Reconfiguring VLSI/WSI Arrays,'' *Intern'l. Conference on Wafer Scale Integration*, pp. 229-235, San Francisco, CA, Jan. 1990.

18. D. Pal and T. Kailath, ''Fast Modified Triangular Factorization of Hermitian Toeplitz and Quasi-Toeplitz Matrices With Arbitrary Rank Profile,'' *1990 IEEE Intern'l. Symp. on Information Theory*, (abstract only) San Diego, January 1990.

Papers Under Review

1. D. Pal and T. Kailath, ''Approximate AR Modeling: An Alternative to Regularization via Schur Algorithm for Matrices with Arbitrary Rank Profile,'' *5th ASSP Workshop on Spectrum Estimation and Modeling.*
2. D. Pal and T. Kailath, ''Fast Modified Triangular Factorization of Hankel, and Related Matrices With Arbitrary Rank Profile,'' *SIAM J. Matrix Analysis and Applications.*
3. D. Pal and T. Kailath, ''Fast Triangular Factorization of Hermitian Toeplitz and Related Matrices With Arbitrary Rank Profile,'' *SIAM Journal on Matrix Analysis.*
4. O. Farotimi, A. Dembo and T. Kailath, ''Optimally Trained Neural Networks,'' *IEEE Trans. on Neural Networks.*
5. T. Kailath and J. Chun, ''Generalized Displacement Structure for Block-Toeplitz, Toeplitz-Block, and Toeplitz-Derived Matrices,'' SIAM J. Matrix Anal. and Appls.
6. T. Kailath, J. Chun and V. Roychowdhury, ''Systolic Array for Solving Toeplitz Systems of Equations,'' *IEEE Trans. ASSP.*
7. V. P. Roychowdhury, L. Thiele, S. K.Rao and T. Kailath, ''On the Localization of Algorithms for VLSI Processor Arrays,'' *IEEE Trans. Computers.*

5 Scientific Personnel Supported By This Project and Degrees Awarded During The Period 1987--1990:

Professor T. Kailath

Graduate Students: S. Balemi, J-H. Chun (Ph.D. 6/89), O. Farotimi, M. Genossar, M. Goldburg, D. Pal (Ph.D. 5/90), V. Roychowdhury (Ph.D. 12/88), T. Varvarigou

Research Associates: A. Dembo, P. G. Gulak, R. Roy, V. Roychowdhury